

---

# **cepton\_sdk Documentation**

**Cepton Technologies**

**Sep 19, 2019**



# CONTENTS

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Samples</b>	<b>3</b>
<b>3</b>	<b>Errors</b>	<b>17</b>
<b>4</b>	<b>Setup</b>	<b>21</b>
<b>5</b>	<b>Sensors</b>	<b>25</b>
<b>6</b>	<b>Points</b>	<b>29</b>
<b>7</b>	<b>Serial</b>	<b>31</b>
<b>8</b>	<b>Networking</b>	<b>33</b>
<b>9</b>	<b>Capture Replay</b>	<b>35</b>
<b>10</b>	<b>C++</b>	<b>37</b>
<b>11</b>	<b>Utilities</b>	<b>39</b>
	<b>Index</b>	<b>45</b>



## OVERVIEW

The Cepton SDK provides the following features

- **Parsing:** parse sensor packets
- **Calibration:** apply sensor calibration
- **Networking:** listen for sensor packets
- **Capture Replay:** read sensor packets from a PCAP file
- **Frame Accumulation:** accumulate and output sensor points by frame

Currently, the Cepton LiDAR packet formats are under active development, and are not publicly available. The SDK is required for **Parsing** and **Calibration**. All other SDK features are optional, and can be done manually by the user.

### 1.1 Getting Started

To start, take a look at [Samples](#). The sample code covers most common use cases.

For prototyping, it is recommended to use the high level C++ API ([C++](#), [Utilities](#)).

### 1.2 Timestamps

All `int64` timestamps are microseconds since the Unix epoch (UTC). All `float` timestamps are time differences measured in seconds. Point timestamps are based on one of the following sources (the first valid source is used):

1. GPS (NMEA + PPS)
2. PTP
3. Host PC

All `float` and `int64` time differences are seconds (measurement period, replay time, frame length, etc.).

### 1.3 Multiple Returns

To enable multiple returns, pass the `CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS` flag during initialization.

The returns are as follows:

1. Strongest signal.

2. Furthest signal, if it is not the strongest. Otherwise, the second strongest signal.

## 1.4 Concurrency

- If networking is enabled, the sdk creates 2 threads for networking. All callbacks occur in these networking threads.
- If networking is disabled, the sdk does not create any threads. All callbacks occur in the main thread.
- All sdk getter functions are thread safe, and can be called from callbacks. Other sdk functions are not guaranteed to be thread safe, and can cause deadlock if called from callbacks.

## 1.5 Minimal SDK

If desired, the following SDK features can be disabled in the SDK and performed manually by the user:

- Networking: *Network*.
- Capture Replay: *Replay*.
- Frame Accumulation: *Frame*.

## 1.6 Sensor Fusion

Cepton sensors do not have a concept of a frame boundaries. This is especially useful with multiple sensors, since there is no need to worry about frame synchronization between sensors. We recommend choosing a desired frame length, and collecting data for all sensors simultaneously (*Process Single*, *Process Multi*). Longer frame lengths will allow for higher point density per frame, but higher latency. A frame length of ~0.1s (10Hz) is recommended for most applications.

## SAMPLES

## 2.1 Basic

Listing 1: samples/basic.cpp

```
1  /**
2   * Sample code for general sdk usage.
3   */
4  #include <vector>
5
6  #include <cepton_sdk_api.hpp>
7
8  #include "common.hpp"
9
10 class FramesListener {
11 public:
12     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
13                        const cepton_sdk::SensorImagePoint *c_image_points) {
14         // Get sensor info
15         cepton_sdk::SensorInformation sensor_info;
16         CEPTON_CHECK_ERROR(cepton_sdk::get_sensor_information(handle, sensor_info));
17
18         // Print info
19         if (i_frame < 5) {
20             std::printf("Received %i points from sensor %i\n", (int)n_points,
21                        (int)sensor_info.serial_number);
22         }
23         ++i_frame;
24     }
25
26 private:
27     std::size_t i_frame = 0;
28 };
29
30 int main(int argc, char **argv) {
31     check_help(argc, argv, "cepton_sdk_sample_basic [capture_path]");
32     std::string capture_path;
33     if (argc >= 2) capture_path = argv[1];
34
35     std::printf("Press Ctrl+C to stop\n");
36
37     // Initialize
38     auto options = cepton_sdk::create_options();
```

(continues on next page)

(continued from previous page)

```

39 options.frame.mode = CEPTON_SDK_FRAME_TIMED;
40 options.frame.length = 0.1f;
41 CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
42
43 // Get sensor
44 std::printf("Waiting for sensor to connect...\n");
45 while (cepton_sdk::get_n_sensors() == 0)
46     CEPTON_CHECK_ERROR(cepton_sdk::api::wait(0.1f));
47 cepton_sdk::SensorInformation sensor_info;
48 CEPTON_CHECK_ERROR(
49     cepton_sdk::get_sensor_information_by_index(0, sensor_info));
50 std::printf("Sensor: %i\n", (int)sensor_info.serial_number);
51
52 // Listen for frames
53 std::printf("Listening for frames...\n");
54 cepton_sdk::api::SensorImageFrameCallback callback;
55 CEPTON_CHECK_ERROR(callback.initialize());
56 FramesListener frames_listener;
57 CEPTON_CHECK_ERROR(
58     callback.listen(&frames_listener, &FramesListener::on_image_frame));
59
60 // Run
61 CEPTON_CHECK_ERROR(cepton_sdk::api::wait());
62
63 // Deinitialize
64 cepton_sdk::deinitialize().ignore();
65 }

```

## 2.2 C Basic

Listing 2: samples/c\_basic.c

```

1  /**
2   * Sample code for general C sdk usage.
3   */
4  #include <stdio.h>
5  #include <time.h>
6
7  #include <cepton_sdk.h>
8
9  void check_sdk_error() {
10     const char *error_msg;
11     const auto error_code = cepton_sdk_get_error(&error_msg);
12     printf("%s: %s\n", cepton_get_error_code_name(error_code), error_msg);
13     exit(1);
14 }
15
16 int frames_got = 0;
17 time_t first_frame_time = 0;
18 time_t current_frame_time = 0;
19
20 void image_frame_callback(CeptonSensorHandle handle, size_t n_points,
21                          const struct CeptonSensorImagePoint *c_points,
22                          void *user_data) {

```

(continues on next page)



(continued from previous page)

```

23  time_t t = time(NULL);
24
25  if (frames_got == 0) first_frame_time = t;
26  frames_got++;
27  if (frames_got < 50)
28      printf("Frame: %4d Time: %lld\n", frames_got, (long long)t);
29  else
30      printf("Frame: %4d Time: %lld Frame rate: %.1fHz\n", frames_got,
31              (long long)t, frames_got * 1.0 / (t - first_frame_time));
32  }
33
34  int main() {
35      // Initialize
36      struct CeptonSDKOptions options = cepton_sdk_create_options();
37      options.frame.mode = CEPTON_SDK_FRAME_TIMED;
38      options.frame.length = 0.1f;
39      cepton_sdk_initialize(CEPTON_SDK_VERSION, &options, NULL, NULL);
40      check_sdk_error();
41
42      // Listen for frames
43      cepton_sdk_listen_image_frames(image_frame_callback, NULL);
44      check_sdk_error();
45
46      // Run
47      while (frames_got < 100)
48          ; // Just spin loop for lack of cross platform sleep in C
49
50      // Deinitialize
51      cepton_sdk_deinitialize();
52      check_sdk_error();
53  }

```

## 2.3 Callback

Listing 3: samples/callback.cpp

```

1  /**
2   * Sample code for callback usage.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  // Sample global callback.
7  void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
8                      const cepton_sdk::SensorImagePoint *c_image_points) {}
9
10 // Sample member callback.
11 class FramesListener {
12 public:
13     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
14                         const cepton_sdk::SensorImagePoint *c_image_points) {}
15 };
16
17 int main(int argc, char **argv) {
18     // Initialize

```

(continues on next page)

(continued from previous page)

```

19  CEPTON_CHECK_ERROR(cepton_sdk::api::initialize());
20  cepton_sdk::api::SensorImageFrameCallback callback;
21  CEPTON_CHECK_ERROR(callback.initialize());
22
23  // Listen lambda
24  CEPTON_CHECK_ERROR(callback.listen(
25      [](cepton_sdk::SensorHandle handle, std::size_t n_points,
26          const cepton_sdk::SensorImagePoint *c_image_points) {}));
27
28  // Listen global function
29  CEPTON_CHECK_ERROR(callback.listen(on_image_frame));
30
31  // Listen member function
32  FramesListener frames_listener;
33  CEPTON_CHECK_ERROR(
34      callback.listen(&frames_listener, &FramesListener::on_image_frame));
35
36  // Deinitialize
37  cepton_sdk::deinitialize().ignore();
38  }

```

## 2.4 Error

Listing 4: samples/error.cpp

```

1  /**
2   * Sample code for error callback usage.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  int main(int argc, char** argv) {
7      // Initialize
8      cepton_sdk::api::SensorErrorCallback error_callback;
9      CEPTON_CHECK_ERROR(error_callback.listen(
10         [&](cepton_sdk::SensorHandle handle,
11             const cepton_sdk::SensorError& error) { throw error; }));
12      CEPTON_CHECK_ERROR(cepton_sdk::initialize(
13          CEPTON_SDK_VERSION, cepton_sdk::create_options(),
14          error_callback.global_on_callback, &error_callback));
15
16      // Deinitialize
17      cepton_sdk::deinitialize().ignore();
18  }

```

## 2.5 Sensor

Listing 5: samples/sensor.cpp

```

1  /**
2   * Sample code for sensor information.
3   */

```

(continues on next page)

(continued from previous page)

```

4  #include <string>
5
6  #include <cepton_sdk_api.hpp>
7
8  #include "common.hpp"
9
10 int main(int argc, char **argv) {
11     check_help(argc, argv, "cepton_sdk_sample_sensor [capture_path]",
12               "Print information for all sensors.");
13     std::string capture_path;
14     if (argc >= 2) capture_path = argv[1];
15
16     // Initialize
17     auto options = cepton_sdk::create_options();
18     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path, true));
19
20     // Get all sensors
21     const int n_sensors = (int)cepton_sdk::get_n_sensors();
22     for (int i = 0; i < n_sensors; ++i) {
23         cepton_sdk::SensorInformation sensor_info;
24         CEPTON_CHECK_ERROR(
25             cepton_sdk::get_sensor_information_by_index(i, sensor_info));
26         std::printf("%i: %s\n", (int)sensor_info.serial_number,
27                    sensor_info.model_name);
28     }
29
30     // Deinitialize
31     cepton_sdk::deinitialize().ignore();
32 }

```

## 2.6 Advanced

### 2.6.1 Frame

Listing 6: samples/advanced/frame.cpp

```

1  /**
2   * Sample code for custom frame accumulation.
3   */
4
5  #include <cepton_sdk_api.hpp>
6
7  int main(int argc, char **argv) {
8     std::string capture_path;
9     if (argc >= 2) capture_path = argv[1];
10
11     // Initialize
12     auto options = cepton_sdk::create_options();
13     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
14     cepton_sdk::api::SensorImageFrameCallback callback;
15     CEPTON_CHECK_ERROR(callback.initialize());
16
17     // Get sensor

```

(continues on next page)

(continued from previous page)

```

18 while (cepton_sdk::get_n_sensors() == 0)
19     CEPTON_CHECK_ERROR(cepton_sdk::api::wait(0.1f));
20 cepton_sdk::SensorInformation sensor_info;
21 CEPTON_CHECK_ERROR(
22     cepton_sdk::get_sensor_information_by_index(0, sensor_info));
23
24 // Create accumulator
25 auto frame_options = cepton_sdk::create_frame_options();
26 frame_options.mode = CEPTON_SDK_FRAME_TIMED;
27 frame_options.length = 0.1f;
28 cepton_sdk::util::FrameAccumulator accumulator(sensor_info);
29 CEPTON_CHECK_ERROR(accumulator.set_options(frame_options));
30 CEPTON_CHECK_ERROR(callback.listen(
31     [&](cepton_sdk::SensorHandle handle, std::size_t n_points,
32         const cepton_sdk::SensorImagePoint *const c_image_points) {
33         if (handle != sensor_info.handle) return;
34         accumulator.add_points((int)n_points, c_image_points);
35     }));
36
37 // Listen
38 CEPTON_CHECK_ERROR(accumulator.callback.listen(
39     [&](int n_points,
40         const cepton_sdk::SensorImagePoint *const c_image_points) {
41         std::printf("Received %i points\n", n_points);
42     }));
43
44 CEPTON_CHECK_ERROR(cepton_sdk::api::wait(5.0f));
45
46 // Deinitialize
47 cepton_sdk::deinitialize().ignore();
48 }

```

## 2.6.2 Network

Listing 7: samples/advanced/network.cpp

```

1 /**
2  * Sample code for custom networking.
3  */
4 #include <asio.hpp>
5
6 #include <cepton_sdk_api.hpp>
7
8 using asio::ip::udp;
9
10 class SocketListener {
11 public:
12     SocketListener() : m_socket(m_io_service, udp::v4()) {
13         m_socket.set_option(asio::socket_base::reuse_address(true));
14         m_socket.bind(udp::endpoint(udp::v4(), 8808));
15     }
16
17     void run() {
18         listen();
19         m_io_service.run_for(std::chrono::seconds(5));

```

(continues on next page)

(continued from previous page)

```

20 }
21
22 void listen() {
23     m_socket.async_receive_from(
24         asio::buffer(m_buffer), m_end_point,
25         [this](const asio::error_code& error, std::size_t buffer_size) {
26             if (buffer_size == 0) return;
27             if (error == asio::error::operation_aborted) return;
28             const CeptonSensorHandle handle =
29                 m_end_point.address().to_v4().to_ulong();
30             // For more accurate timestamps, a separate network receive thread
31             // should be
32             // used.
33             const int64_t timestamp = cepton_sdk::util::get_timestamp_usec();
34             CEPTON_CHECK_ERROR(cepton_sdk::mock_network_receive(
35                 handle, timestamp, m_buffer.data(), buffer_size));
36             listen();
37         });
38 }
39
40 private:
41     asio::io_context m_io_service;
42     udp::socket m_socket;
43     udp::endpoint m_end_point;
44     std::array<uint8_t, 4096> m_buffer;
45 };
46
47 int main() {
48     // Initialize
49     auto options = cepton_sdk::create_options();
50     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
51     options.frame.mode = CEPTON_SDK_FRAME_COVER;
52     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));
53
54     // Listen for points
55     cepton_sdk::api::SensorImageFrameCallback callback;
56     CEPTON_CHECK_ERROR(callback.initialize());
57     CEPTON_CHECK_ERROR(
58         callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
59                         const cepton_sdk::SensorImagePoint* c_image_points) {
60             std::printf("Received %i points from sensor %lli\n", (int)n_points,
61                         (long long)handle);
62         }));
63
64     SocketListener listener;
65     listener.run();
66
67     // Deinitialize
68     cepton_sdk::deinitialize().ignore();
69 }

```

## 2.6.3 Organize Points

Listing 8: samples/advanced/organize\_points.cpp

```

1  /*****
2  **
3  **  Copyright (C) 2019 Cepton Technologies. All Rights Reserved.  **
4  **  Contact: https://www.cepton.com                               **
5  **                                                                **
6  **  Sample code which opens a cepton sensor pcap file, organizes **
7  **  the points and continuously saves the most recent organized **
8  **  points to a frame to a cvs file "organized_cloud.cvs         **
9  **                                                                **
10 *****/
11 #include <cepton_sdk_util.hpp>
12 #include <cepton_sdk/capture.hpp>
13 #include <cepton_sdk_api.hpp>
14
15 using namespace cepton_sdk::util;
16
17 int main(int argc, char** argv) {
18     if (argc < 2) return -1;
19     const std::string capture_path = argv[1];
20
21     // Initialize sdk
22     auto options = cepton_sdk::create_options();
23     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
24     options.frame.mode = CEPTON_SDK_FRAME_COVER;
25     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));
26
27     cepton_sdk::SensorInformation sensor_info;
28
29     OrganizedCloud organized_cloud;
30     std::ofstream os;
31
32     cepton_sdk::Capture m_capture;
33     CEPTON_CHECK_ERROR(m_capture.open_for_read(capture_path));
34
35     CEPTON_CHECK_ERROR(
36         cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
37                                     CEPTON_SDK_CONTROL_DISABLE_NETWORK));
38     CEPTON_CHECK_ERROR(cepton_sdk_clear());
39
40     // Listen for points
41     cepton_sdk::api::SensorImageFrameCallback callback;
42     CEPTON_CHECK_ERROR(callback.initialize());
43     callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
44                        const cepton_sdk::SensorImagePoint* c_image_points) {
45
46         cepton_sdk::get_sensor_information(handle, sensor_info);
47
48         std::printf("Received %i points from sensor %lli\n", static_cast<int>(n_points),
49                    static_cast<long long>(handle));
50
51         Organizer organizer(sensor_info);
52
53         organizer.organize_points(n_points,
54                                 sensor_info.return_count,
55                                 c_image_points,

```

(continues on next page)

(continued from previous page)

```

56         organized_cloud);
57
58     os.open("organize_cloud.csv");
59     for (const auto& point : organized_cloud.points)
60     {
61         if (point.valid)
62         {
63             float x = 0;
64             float y = 0;
65             float z = 0;
66             cepton_sdk::util::convert_image_point_to_point(
67                 point.image_x, point.image_z, point.distance, x,
68                 y, z);
69
70             os << x << "," << y << "," << z << "\n";
71         }
72     }
73     os.close();
74 });
75
76 while (true) {
77     cepton_sdk::Capture::PacketHeader header;
78     const uint8_t* data;
79     CEPTON_CHECK_ERROR(m_capture.next_packet(header, data));
80
81     const cepton_sdk::SensorHandle handle =
82         static_cast<ception_sdk::SensorHandle>(header.ip_v4) |
83         CEPTON_SENSOR_HANDLE_FLAG MOCK;
84     CEPTON_CHECK_ERROR(ception_sdk_mock_network_receive(
85         handle, header.timestamp, data, static_cast<size_t>(header.data_size)));
86 }
87 }

```

## 2.6.4 Process Multi

Listing 9: samples/advanced/process\_multi.cpp

```

1  /**
2   * Sample code for processing multiple sensor data.
3   */
4  #include <string>
5  #include <vector>
6
7  #include <ception_sdk_api.hpp>
8
9  struct Frame {
10     int64_t timestamp;
11     std::map<ception_sdk::SensorHandle, std::vector<ception_sdk::SensorImagePoint>>
12         image_points_dict;
13 };
14
15 class FrameAccumulator {
16 public:
17     void on_image_frame(
18         cepton_sdk::SensorHandle handle, std::size_t n_points,

```

(continues on next page)

(continued from previous page)

```

19     const cepton_sdk::SensorImagePoint* const c_image_points) {
20     cepton_sdk::util::LockGuard lock(m_mutex);
21
22     auto& image_points = m_image_points_dict[handle];
23     image_points.reserve(image_points.size() + n_points);
24     image_points.insert(image_points.end(), c_image_points,
25                         c_image_points + n_points);
26
27     check_and_publish();
28 }
29
30 private:
31 void check_and_publish() {
32     const auto timestamp = cepton_sdk::api::get_time();
33
34     if (timestamp < m_timestamp) return;
35     if ((timestamp - m_timestamp) < int64_t(m_frame_length * 1e6f)) return;
36     m_timestamp = timestamp;
37
38     auto frame = std::make_shared<Frame>();
39     frame->timestamp = timestamp;
40     frame->image_points_dict = m_image_points_dict;
41     m_image_points_dict.clear();
42     queue.push(frame);
43 }
44
45 public:
46     cepton_sdk::util::SingleConsumerQueue<Frame> queue;
47
48 private:
49     std::timed_mutex m_mutex;
50     float m_frame_length = 0.1f;
51     int64_t m_timestamp = 0;
52     std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
53         m_image_points_dict;
54 };
55
56 int main(int argc, char** argv) {
57     std::string capture_path;
58     if (argc >= 2) capture_path = argv[1];
59
60     // Initialize
61     auto options = cepton_sdk::create_options();
62     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
63     cepton_sdk::api::SensorImageFrameCallback callback;
64     CEPTON_CHECK_ERROR(callback.initialize());
65     if (cepton_sdk::capture_replay::is_open())
66         CEPTON_CHECK_ERROR(cepton_sdk::capture_replay::resume());
67
68     // Listen
69     FrameAccumulator accumulator;
70     CEPTON_CHECK_ERROR(
71         callback.listen(&accumulator, &FrameAccumulator::on_image_frame));
72     while (true) {
73         const auto frame = accumulator.queue.pop(0.01f);
74         if (!frame) continue;
75         // Do processing

```

(continues on next page)



(continued from previous page)

```

76     }
77
78     // Deinitialize
79     cepton_sdk::deinitialize().ignore();
80 }

```

## 2.6.5 Process Single

Listing 10: samples/advanced/process\_single.cpp

```

1  /**
2   * Sample code for processing single sensor data.
3   */
4  #include <memory>
5  #include <vector>
6
7  #include <cepton_sdk_api.hpp>
8
9  struct Frame {
10     int64_t timestamp;
11     cepton_sdk::SensorHandle handle;
12     std::vector<cepton_sdk::SensorImagePoint> image_points;
13 };
14
15 int main(int argc, char **argv) {
16     std::string capture_path;
17     if (argc >= 2) capture_path = argv[1];
18
19     // Initialize
20     auto options = cepton_sdk::create_options();
21     options.frame.mode = CEPTON_SDK_FRAME_TIMED;
22     options.frame.length = 0.1f;
23     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
24     cepton_sdk::api::SensorImageFrameCallback callback;
25     CEPTON_CHECK_ERROR(callback.initialize());
26     if (cepton_sdk::capture_replay::is_open())
27         CEPTON_CHECK_ERROR(cepton_sdk::capture_replay::resume());
28
29     // Listen
30     cepton_sdk::util::SingleConsumerQueue<Frame> queue;
31     CEPTON_CHECK_ERROR(
32         callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
33                             const cepton_sdk::SensorImagePoint *c_image_points) {
34             auto frame = std::make_shared<Frame>();
35             frame->timestamp = cepton_sdk::api::get_time();
36             frame->handle = handle;
37             frame->image_points.reserve(n_points);
38             frame->image_points.insert(frame->image_points.end(), c_image_points,
39                                     c_image_points + n_points);
40             queue.push(frame);
41         }));
42     while (true) {
43         const auto frame = queue.pop(0.01f);
44         if (!frame) continue;
45         // Do processing

```

(continues on next page)

(continued from previous page)

```

46     }
47
48     // Deinitialize
49     cepton_sdk::deinitialize().ignore();
50 }

```

## 2.6.6 Replay

Listing 11: samples/advanced/replay.cpp

```

1  /**
2   * Sample code for custom packet replaying.
3   */
4  #include <cepton_sdk/capture.hpp>
5  #include <cepton_sdk_api.hpp>
6
7  class CaptureReplay {
8  public:
9      CaptureReplay(const std::string& path) {
10         CEPTON_CHECK_ERROR(m_capture.open_for_read(path));
11
12         CEPTON_CHECK_ERROR(
13             cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
14             CEPTON_SDK_CONTROL_DISABLE_NETWORK));
15         CEPTON_CHECK_ERROR(cepton_sdk_clear());
16     }
17
18     ~CaptureReplay() {
19         m_capture.close();
20         if (cepton_sdk_is_initialized()) {
21             CEPTON_CHECK_ERROR(cepton_sdk_clear());
22         }
23     }
24
25     void run() {
26         while (true) {
27             cepton_sdk::Capture::PacketHeader header;
28             const uint8_t* data;
29             CEPTON_CHECK_ERROR(m_capture.next_packet(header, data));
30
31             const cepton_sdk::SensorHandle handle =
32                 (cepton_sdk::SensorHandle)header.ip_v4 |
33                 CEPTON_SENSOR_HANDLE_FLAG MOCK;
34             CEPTON_CHECK_ERROR(cepton_sdk_mock_network_receive(
35                 handle, header.timestamp, data, header.data_size));
36         }
37     }
38
39 private:
40     cepton_sdk::Capture m_capture;
41 };
42
43 int main(int argc, char** argv) {
44     if (argc < 2) return -1;
45     const std::string capture_path = argv[1];

```

(continues on next page)

(continued from previous page)

```

46
47 // Initialize
48 auto options = cepton_sdk::create_options();
49 options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
50 options.frame.mode = CEPTON_SDK_FRAME_COVER;
51 CEPTON_CHECK_ERROR(ception_sdk::api::initialize(options));
52
53 // Listen for points
54 cepton_sdk::api::SensorImageFrameCallback callback;
55 CEPTON_CHECK_ERROR(callback.initialize());
56 CEPTON_CHECK_ERROR(
57     callback.listen([](ception_sdk::SensorHandle handle, std::size_t n_points,
58                       const cepton_sdk::SensorImagePoint* c_image_points) {
59         std::printf("Received %i points from sensor %lli\n", (int)n_points,
60                     (long long)handle);
61     }));
62
63 // Run
64 CaptureReplay replay(capture_path);
65 replay.run();
66
67 // Deinitialize
68 cepton_sdk::deinitialize().ignore();
69 }

```

## 2.6.7 Stray

Listing 12: samples/advanced/stray.cpp

```

1 /**
2  * Sample code for stray filter usage.
3  */
4 #include <vector>
5
6 #include <ception_sdk_api.hpp>
7
8 int main(int argc, char **argv) {
9     // Initialize
10    CEPTON_CHECK_ERROR(ception_sdk::api::initialize());
11
12    cepton_sdk::api::SensorImageFrameCallback callback;
13    CEPTON_CHECK_ERROR(callback.initialize());
14    cepton_sdk::util::StrayFilter filter;
15    CEPTON_CHECK_ERROR(
16        callback.listen([&](ception_sdk::SensorHandle handle, std::size_t n_points,
17                            const cepton_sdk::SensorImagePoint *c_image_points) {
18            // Get sensor
19            cepton_sdk::SensorInformation sensor_info;
20            CEPTON_CHECK_ERROR(
21                cepton_sdk::get_sensor_information(handle, sensor_info));
22
23            // Copy points to buffer
24            std::vector<ception_sdk::SensorImagePoint> image_points;
25            image_points.insert(image_points.begin(), c_image_points,
26                               c_image_points + n_points);

```

(continues on next page)

(continued from previous page)

```
27
28     // Filter stray
29     filter.init(sensor_info);
30     filter.run((int)n_points, image_points.data());
31     });
32
33     // Run
34     CEPTON_CHECK_ERROR(cepton_sdk::api::wait(5.0f));
35
36     // Deinitialize
37     cepton_sdk::deinitialize().ignore();
38 }
```

## ERRORS

### 3.1 Types

**class SensorError : public runtime\_error**

Error returned by most functions.

Implicitly convertible from/to SensorErrorCode. Getter functions do not return an error, because they cannot fail. Will call CEPTON\_RUNTIME\_ASSERT if nonzero error is not used (call `ignore` to manually use error).

#### Public Functions

**SensorError** (SensorErrorCode *code\_*, **const** std::string &*msg\_*)

**SensorError** (SensorErrorCode *code\_*)

**SensorError** ()

**~SensorError** ()

**SensorError** (**const** *SensorError* &*other*)

*SensorError* &**operator=** (**const** *SensorError* &*other*)

bool **used** () **const**  
Internal use only.

**const** *SensorError* &**ignore** () **const**  
Mark error as used.

**const** std::string &**msg** () **const**  
Returns error message;.

SensorErrorCode **code** () **const**  
Returns error code.

**operator** SensorErrorCode () **const**  
Implicitly convert to SensorErrorCode.

**operator** bool () **const**  
Returns `false` if code is CEPTON\_SUCCESS, true otherwise.

**const** std::string **name** () **const**

bool **is\_error** () **const**

```
bool is_fault () const
_CeptonSensorErrorCode
    Values:
    0
    1
    2
    4
    5
    6
    Networking error.
    7
    8
    Invalid value or uninitialized struct.
    9
    10
    11
    12
    13
    14
    15
    1000
    Internal parameter out of range.
    1001
    Reading exceed spec.
    1002
    Reading exceeds spec.
    1003
    1004
    1005
    1006
    1007
    1008
typedef int32_t CeptonSensorErrorCode
```

## 3.2 Methods

```
const char* cepton_get_error_code_name (CeptonSensorErrorCode error_code)
    Returns string name of error code.
    Returns empty string if error code is invalid.
```

int **cepton\_is\_error\_code** (*CeptonSensorErrorCode* error\_code)

Returns true if error name is of the form CEPTON\_ERROR\_\*, false otherwise.

int **cepton\_is\_fault\_code** (*CeptonSensorErrorCode* error\_code)

Returns true if error name is of the form CEPTON\_FAULT\_\*, false otherwise.

*CeptonSensorErrorCode* **cepton\_sdk\_get\_error** (const char \*\* error\_msg)

Returns and clears last sdk error.

error\_msg is owned by the SDK, and is valid until the next call in the current thread.





## 4.1 Types

### `_CeptonSDKControl`

SDK control flags.

*Values:*

- 1  
Disable networking operations.  
Useful for running multiple instances of sdk in different processes. Must pass packets manually to `cepton_sdk::mock_network_receive`.
- 2  
Disable marking image clipped points as invalid.  
Does not affect number of points returned.
- 3  
Disable marking distance clipped points as invalid.  
Does not affect number of points returned.
- 4  
Enable multiple returns.  
When set, `cepton_sdk::SensorInformation::return_count` will indicate the number of returns per laser. Can only be set at sdk initialization.
- 5  
Enable marking stray points as invalid (measurement noise).  
Uses `cepton_sdk::util::StrayFilter` to mark points invalid.  
Does not affect number of points returned.
- 6  
Always use packet timestamps (disable GPS/PTP timestamps).
- 7  
Enable marking crosstalk points as invalid.

**`typedef uint32_t CeptonSDKControl`**

### `_CeptonSDKFrameMode`

Controls frequency of points being reported.

*Values:*

- 0 Report points by packet.
- 1 Report points at fixed time intervals.  
Interval controlled by *CeptonSDKFrameOptions::length*.
- 2 Report points when the field of view is covered once.
  - For HR80 series, detects half scan cycle (left-to-right or right-to-left).
- 3 Report points when the scan pattern goes through a full cycle.  
Typically 2x longer frame than COVER mode.
  - For HR80 series, detects full scan cycle (left-to-right-to-left).
  - For VISTA series, internally uses TIMED mode.

```
typedef uint32_t CeptonSDKFrameMode
struct CeptonSDKFrameOptions
```

### Public Members

size\_t **signature**  
Internal use only.

*CeptonSDKFrameMode* **mode**  
Default: CEPTON\_SDK\_FRAME\_STREAMING.

float **length**  
Frame length [seconds].  
Default: 0.05. Only used if mode=CEPTON\_SDK\_FRAME\_TIMED.

struct *CeptonSDKFrameOptions* **cepton\_sdk\_create\_frame\_options()**  
Create default frame options.

**struct CeptonSDKOptions**  
SDK initialization options.

### Public Members

size\_t **signature**  
Internal use only.

*CeptonSDKControl* **control\_flags**  
Default: 0.

struct *CeptonSDKFrameOptions* **frame**

uint16\_t **port**  
Default: 8808.

struct *CeptonSDKOptions* **cepton\_sdk\_create\_options()**  
Create default options.

```
typedef void (*cepton_sdk::FpSensorErrorCallback) (SensorHandle handle, SensorErrorCode
error_code, const char *error_msg,
const void *error_data, size_t error_data_size, void *user_data)
```

## 4.2 Methods

*CeptonSensorErrorCode* **cepton\_sdk\_initialize** (int ver, **const** struct *CeptonSDKOptions* \***const options**, FpCeptonSensorErrorCallback cb, void \***const user\_data**)

Initializes settings and networking.

Must be called before any other sdk function listed below.

*CeptonSensorErrorCode* **cepton\_sdk\_deinitialize** ()  
Resets everything and deallocates memory.

*CeptonSensorErrorCode* **cepton\_sdk\_clear** ()  
Clears sensors.

Use when loading/unloading capture file.

*CeptonSensorErrorCode* **cepton\_sdk\_set\_control\_flags** (*CeptonSDKControl* mask, *CeptonSDKControl* flags)

*CeptonSDKControl* **cepton\_sdk\_get\_control\_flags** ()

int **cepton\_sdk\_has\_control\_flag** (*CeptonSDKControl* flag)

uint16\_t **cepton\_sdk\_get\_port** ()

*CeptonSensorErrorCode* **cepton\_sdk\_set\_port** (uint16\_t port)  
Sets network listen port.

Default: 8808.

*CeptonSensorErrorCode* **cepton\_sdk\_set\_frame\_options** (**const** struct *CeptonSDKFrameOptions* \***const options**)

*CeptonSDKFrameMode* **cepton\_sdk\_get\_frame\_mode** ()

float **cepton\_sdk\_get\_frame\_length** ()



## **SENSORS**

### **5.1 Types**

```
typedef CeptonSensorHandle cepton_sdk::SensorHandle
```

```
_CeptonSensorModel
```

*Values:*

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

18

```
typedef uint16_t CeptonSensorModel
```

```
struct CeptonSensorInformation
```

## Public Members

CeptonSensorHandle **handle**

uint64\_t **serial\_number**

char CeptonSensorInformation::model\_name[28]

*CeptonSensorModel* **model**

uint16\_t **reserved**

char CeptonSensorInformation::firmware\_version[28]

uint8\_t **major**

uint8\_t **minor**

uint8\_t CeptonSensorInformation::unused[2]

struct *CeptonSensorInformation*::@16 **formal\_firmware\_version**

float **last\_reported\_temperature**  
[celsius]

float **last\_reported\_humidity**  
[%]

float **last\_reported\_age**  
[hours]

float **measurement\_period**  
Time between measurements [seconds].

int64\_t **ptp\_ts**  
[microseconds]

uint8\_t **gps\_ts\_year**  
0-99 (2017 -> 17)

uint8\_t **gps\_ts\_month**  
1-12

uint8\_t **gps\_ts\_day**  
1-31

uint8\_t **gps\_ts\_hour**  
0-23

uint8\_t **gps\_ts\_min**  
0-59

uint8\_t **gps\_ts\_sec**  
0-59

uint8\_t **return\_count**

uint8\_t **segment\_count**  
Number of image segments.

uint32\_t **flags**  
Bit flags.

uint32\_t **is\_mocked**  
Created by capture replay.

```

uint32_t is_pps_connected
    GPS PPS is available.

uint32_t is_nmea_connected
    GPS NMEA is available.

uint32_t is_ptp_connected
    PTP is available.

uint32_t is_calibrated

uint32_t is_over_heated
    Hit temperature limit (only available in Vista Gen2 for now)

uint32_t is_sync_firing_enabled

union CeptonSensorInformation::@17 CeptonSensorInformation::@18

```

## 5.2 Methods

```

size_t cepton_sdk_get_n_sensors ()
    Get number of sensors attached. Use to check for new sensors. Sensors are not deleted until deinitialization.

CeptonSensorErrorCode cepton_sdk_get_sensor_handle_by_serial_number (uint64_t se-
                                                                    rial_number,
                                                                    CeptonSen-
                                                                    sorHandle
                                                                    *const handle)

    Returns error if sensor not found.

CeptonSensorErrorCode cepton_sdk_get_sensor_information_by_index (size_t idx, struct
                                                                    CeptonSensorInfor-
                                                                    mation *const info)

    Valid indices are in range [0, n_sensors). Returns error if index invalid.

CeptonSensorErrorCode cepton_sdk_get_sensor_information (CeptonSensorHandle handle,
                                                                    struct CeptonSensorInformation
                                                                    *const info)

    Returns error if sensor not found.

```





## 6.1 Types

### **struct CeptonSensorImagePoint**

Point in image coordinates (focal length = 1).

To convert to 3d point, refer to `cepton_sdk_util.hpp`.

#### **Public Members**

`int64_t timestamp`

Unix time [microseconds].

`float image_x`

x image coordinate.

`float distance`

Distance [meters].

`float image_z`

z image coordinate.

`float intensity`

Diffuse reflectance.

`CeptonSensorReturnType return_type`

`uint8_t flags`

Bit flags.

`uint8_t valid`

If `false`, then the distance and intensity are invalid.

`uint8_t saturated`

If `true`, then the intensity is invalid. Also, the distance is valid, but inaccurate.

`union CeptonSensorImagePoint::@21 CeptonSensorImagePoint::@22`

`uint8_t CeptonSensorImagePoint::reserved[2]`

## 6.2 Methods

```
typedef void (* FpCeptonSensorImageDataCallback) (CeptonSensorHandle handle,
                                                    size_t n_points, const struct Cep-
                                                    tonSensorImagePoint *c_points,
                                                    void *user_data)
```

Callback for receiving image points.

Set the frame length to control the callback rate.

```
CeptonSensorErrorCode cepton_sdk_listen_image_frames (FpCeptonSensorImageDataCallback cb,
                                                    void *const user_data)
```

Sets image frames callback.

Returns points at frequency specified by `cepton_sdk::FrameOptions::mode`. Each frame contains all possible points (use `cepton_sdk::SensorImagePoint::valid` to filter points). Points are ordered by measurement, segment, and return:

```
measurement_count = n_points / (segment_count * return_count)
idx = ((i_measurement) * segment_count + i_segment) * return_count + i_return
```

Returns error if callback already registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_image_frames ()
```

## SERIAL

Serial callback. Primarily used for receiving data from GPS/INS attached to sensor.

### 7.1 Types

```
typedef void (* FpCeptonSerialReceiveCallback) (CeptonSensorHandle handle, const  
char *str, void *user_data)
```

Callback for receiving serial data (e.g. NMEA).

### 7.2 Methods

```
CeptonSensorErrorCode cepton_sdk_listen_serial_lines (FpCeptonSerialReceiveCallback cb,  
void *const user_data)
```

Sets serial line callback.

Useful for listening to NMEA data from GPS attached to sensor.

Each callback contains 1 line of serial data (including newline characters).

Returns error if callback already registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_serial_lines ()
```



## NETWORKING

Network callback for debugging.

### 8.1 Types

```
typedef void (* FpCeptonNetworkReceiveCallback) (CeptonSensorHandle handle,  
int64_t timestamp, const uint8_t *buffer,  
size_t buffer_size, void *user_data)
```

Callback for receiving network packets.

#### Parameters

- *handle*: Unique sensor identifier (e.g. IP address). Returns error if callback already set.

### 8.2 Methods

```
CeptonSensorErrorCode cepton_sdk_listen_network_packet (FpCeptonNetworkReceiveCallback cb,  
void *const user_data)
```

Sets network packets callback.

Only 1 callback can be registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_network_packet ()
```



## CAPTURE REPLAY

PCAP capture file replay. Source code can be found in the *source* folder. Functions are not thread safe, and should only be called from the main thread.

int **cepton\_sdk\_capture\_replay\_is\_open** ()

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_open** (const char \*const *path*)

Opens capture file.

Must be called before any other replay functions listed below.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_close** ()

const char\* **cepton\_sdk\_capture\_replay\_get\_filename** ()

int64\_t **cepton\_sdk\_capture\_replay\_get\_start\_time** ()

Returns capture start timestamp (unix time [microseconds]).

float **cepton\_sdk\_capture\_replay\_get\_position** ()

Returns capture file position [seconds].

float **cepton\_sdk\_capture\_replay\_get\_length** ()

Returns capture file length [seconds].

int **cepton\_sdk\_capture\_replay\_is\_end** ()

Returns true if at end of capture file.

This is only relevant when using *resume\_blocking* methods.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_seek** (float *position*)

Seek to capture file position [seconds].

Position must be in range [0.0, capture length). Returns error if position is invalid.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_set\_enable\_loop** (int *enable\_loop*)

If enabled, replay will automatically rewind at end.

int **cepton\_sdk\_capture\_replay\_get\_enable\_loop** ()

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_set\_speed** (float *speed*)

Replay speed multiplier for asynchronous replay.

float **cepton\_sdk\_capture\_replay\_get\_speed** ()

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_resume\_blocking\_once** ()

Replay next packet in current thread without sleeping.

Pauses replay thread if it is running.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_resume\_blocking** (float *duration*)

Replay multiple packets synchronously.

No sleep between packets. Resume duration must be non-negative. Pauses replay thread if it is running.

int **cepton\_sdk\_capture\_replay\_is\_running** ()

Returns true if replay thread is running.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_resume** ()

Packets are replayed in realtime. Replay thread sleeps in between packets.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_pause** ()

Pauses asynchronous replay thread.



High level C++ API for prototyping (*cepton\_sdk\_api.hpp*). Methods are agnostic to live/replay mode.

`bool cepton_sdk::api::is_live()`  
Returns true if capture replay is not open.

`bool cepton_sdk::api::is_end()`

`int64_t cepton_sdk::api::get_time()`  
Returns capture replay time or live time.

*SensorError* `cepton_sdk::api::wait` (float *t\_length* = -1.0f)  
Sleeps or resumes capture replay for duration.  
If *t\_length* < 0, then waits forever.

## 10.1 Errors

*SensorError* `cepton_sdk::api::check_error` (const *SensorError* &*error*, const std::string &*msg* = "")  
DEPRECATED: use CEPTON\_CHECK\_ERROR.

*SensorError* `cepton_sdk::api::log_error` (const *SensorError* &*error*, const std::string &*msg* = "")  
DEPRECATED: use CEPTON\_LOG\_ERROR.

`void cepton_sdk::api::default_on_error` (*SensorHandle* *h*, SensorErrorCode *error\_code*, const char \*const *error\_msg*, const void \*const *error\_data*, std::size\_t *error\_data\_size*, void \*const *instance*)

Basic SDK error callback.

Calls `cepton_sdk::api::check_error_code`.

## 10.2 Setup

*SensorError* `cepton_sdk::api::initialize` (Options *options* = `create_options()`, const std::string &*capture\_path* = "", bool *enable\_wait* = false)

Initialize SDK and optionally starts capture replay.

If *enable\_wait* is true, waits a few seconds to initialize sensors.

`class SensorErrorCallback : public cepton_sdk::util::Callback<SensorHandle, const SensorError>`  
Callback for sensor errors.

## Public Static Functions

```
static void global_on_callback(SensorHandle handle, SensorErrorCode error_code, const
                             char *error_msg, const void *const error_data, size_t er-
                             ror_data_size, void *const instance)
```

**class SensorImageFrameCallback** : public cepton\_sdk::util::Callback<*SensorHandle*, std::size\_t, const SensorImagePo  
 Callback for image frames.  
 Must call `initialize` before use.

## Public Functions

```
~SensorImageFrameCallback()
```

```
SensorError initialize()
```

```
SensorError deinitialize()
```

```
bool is_initialized() const
```

**class NetworkPacketCallback** : public cepton\_sdk::util::Callback<*SensorHandle*, int64\_t, uint8\_t const \*, std::size\_t>  
 Callback for network packets.  
 Must call `initialize` before use.

## Public Functions

```
~NetworkPacketCallback()
```

```
SensorError initialize()
```

```
SensorError deinitialize()
```

## 10.3 Sensors

```
bool cepton_sdk::api::has_sensor_by_serial_number(uint64_t serial_number)
```

```
SensorError cepton_sdk::api::get_sensor_information_by_serial_number(uint64_t se-
                                                                    rial_number,
                                                                    SensorIn-
                                                                    formation
                                                                    &info)
```

Returns error if sensor not found.

```
std::vector<uint64_t> cepton_sdk::api::get_sensor_serial_numbers()
```

Returns serial numbers for all sensors.

## UTILITIES

Utility functions and classes for prototyping (*cepton\_sdk\_util.hpp*).

### 11.1 Common

`int64_t cepton_sdk::util::get_timestamp_usec()`

Returns current unix timestamp [microseconds].

This is the timestamp format used by all sdk functions.

### 11.2 Points

`void cepton_sdk::util::convert_image_point_to_point` (float *image\_x*, float *image\_z*, float *distance*, float &*x*, float &*y*, float &*z*)

Convert image point to 3d point.

**struct SensorPoint**

3d point class.

Can't subclass from SensorImagePoint, needs to be POD.

#### Public Members

`int64_t timestamp`  
Unix time [microseconds].

`float image_x`  
x image coordinate.

`float distance`  
Distance [meters].

`float image_z`  
z image coordinate.

`float intensity`  
Diffuse reflectance.

`CeptonSensorReturnType return_type`

`uint8_t flags`

```

uint8_t valid
uint8_t saturated
union cepton_sdk::util::SensorPoint::[anonymous] [anonymous]
uint8_t reserved[5]
float x
    x cartesian coordinate
float y
    y cartesian coordinate
float z
    z cartesian coordinate
void cepton_sdk::util::convert_sensor_image_point_to_point (const    SensorImage-
                                                             Point    &image_point,
                                                             SensorPoint &point)
Convenience    method    to    convert    cepton_sdk::SensorImagePoint    to
cepton_sdk::SensorPoint.

```

## 11.3 Callbacks

```
template<typename ...TArgs>
```

```
class Callback
```

Expands SDK callback functionality.

Allows for multiple callbacks to be registered. Allows for registering lambdas and member functions. See `samples/basic.cpp`.

### Public Functions

```
void clear ()
    Clear all listeners.
```

```
SensorError listen (const std::function<void> TArgs...
    > &func, uint64_t *const id = nullptr) Register std::function.
```

```
template<typename TClass>
SensorError listen (TClass *const instance, MemberFunction<TClass, TArgs...> func, uint64_t
    *const id = nullptr)
    Register instance member function.
```

```
void operator () (TArgs... args) const
    Emit callback.
```

### Public Static Functions

```
static void global_on_callback (TArgs... args, void *const instance)
    Used for registering as c callback.
```

## 11.4 Frames

### class **FrameDetector**

Detects frames in streaming sensor data.

#### Public Functions

**FrameDetector** (**const** SensorInformation &*sensor\_info*)

**const** FrameOptions &**get\_options** () **const**

*SensorError* **set\_options** (**const** FrameOptions &*options*)

void **reset** ()

Completely resets detector.

Only use if also clearing points accumulator.

bool **add\_point** (**const** SensorImagePoint &*point*)

Returns true if frame found.

Automatically resets after frame is found.

#### Public Members

bool **frame\_found**

int **frame\_idx**

float **frame\_x**

Number of points in current frame.

### class **FrameAccumulator**

Accumulates image points, and emits frames to callback.

See `samples/frame.cpp`.

#### Public Functions

**FrameAccumulator** (**const** SensorInformation &*sensor\_info*)

FrameOptions **get\_options** () **const**

*SensorError* **set\_options** (**const** FrameOptions &*options*)

void **clear** ()

void **add\_points** (int *n\_points*, **const** SensorImagePoint \***const** *image\_points*)

#### Public Members

Callback<int, **const** SensorImagePoint \*> **callback**

## 11.5 Organizer

### **struct OrganizedCloud**

The *OrganizedCloud* struct An organized version of the cepton point cloud.

#### **Public Functions**

int **getIndex** (int *row*, int *col*, int *n\_return*)

getIndex Returns the index of the point corresponding to the inputted row, col and return number.

#### **Return**

#### **Parameters**

- [in] *row*: Row index
- [in] *col*: Col index
- [in] *n\_return*: Return index

#### **Public Members**

int64\_t **timestamp\_start**

timestamp\_start The time of the oldest point in the cloud

int64\_t **timestamp\_end**

timestamp\_end The time of the newest point in the cloud

int **height**

height Height of the cloud. Represents how many rows there are in the cloud

int **width**

width Width of the cloud. Represents how many columns there are in the cloud

int **n\_returns**

n\_returns Number of return represented by the cloud.

std::vector<*CellInfo*> **info\_cells**

info\_cells Vector of cell info which provide information about the matching points

std::vector<CeptonSensorImagePoint> **points**

points Vector of organized points. Stored in Return, Row, Col order. So to get a point at row 10, col 15, return 1 would be points[(row \* width

- col) n\_returns + return

### **struct CellInfo**

The *CellInfo* struct.

#### **Public Members**

bool **occupied\_cell** = false

occupied\_cell Is the cell at this index occupied with a point. If false can't assume this represents free space.

```
int original_index = -1
```

**original\_index** Index of the point that was used to generate the organized point. Can be used to match back with original data if required. Should only be used if **occupied\_cell** is true.

## **class Organizer**

The *Organizer* class Performs organization on cepton unorganized points. Creates an angular grid, places each point within that grid and outputs a point for each location in the grid in a row/col format. Thread safe. Defaults to a 0.4deg spaced grid.

## **Public Types**

```
enum OrganizerMode
```

*Values:*

**RECENT**

Output the most recent point from the frame that fell within the grid

**CENTER**

Output the center of the grid. Uses median point distance.

## **Public Functions**

```
Organizer (cepton_sdk::SensorInformation sensor_info)
```

*Organizer.*

### **Parameters**

- **sensor\_info**: Sensor info for organizer. Used to set min/max angles

```
void organize_points (const int num_points_in, const int n_returns, const CeptonSensorImagePoint *const unorganized_points, cepton_sdk::util::OrganizedCloud &organized_points)
```

**organize\_points**

### **Parameters**

- [in] **num\_points\_in**: Number of unorganized points
- [in] **n\_returns**: Number of returns
- [in] **unorganized\_points**: Unorganized points to process
- [out] **organized\_points**: Points in organized form

```
void mode (OrganizerMode mode)
```

**mode**

### **Parameters**

- **mode**: Change the mode of the organizer. [RECENT] Points are the most recent which fill within the grid. [CENTER] Points outputted are at the center of the grid. More even spacing but less accurate.

```
void binSize (float bin_size)
```

**binSize** Change the bin size of the organizer

**Parameters**

- `bin_size`: The horizontal and vertical bin size to set. In radians

void **settings** (*OrganizerSettings* *organizer\_settings*)  
settings

**Parameters**

- `organizer_settings`: Change organizer settings

*Organizer::OrganizerSettings* **settings** ()  
settings

**Return** The settings the organizer is using

**struct** OrganizerSettings

**Public Members**

float **horizontal\_range\_radians** = to\_radians(70.f)

float **vertical\_range\_radians** = to\_radians(30.f)

float **horizontal\_bin\_size\_radians** = to\_radians(0.4f)

float **vertical\_bin\_size\_radians** = to\_radians(0.4f)

*OrganizerMode* **mode** = *OrganizerMode::*RECENT



## INDEX

### C

cepton\_get\_error\_code\_name (*C function*), 18  
 cepton\_is\_error\_code (*C function*), 18  
 cepton\_is\_fault\_code (*C function*), 19  
 cepton\_sdk::api::check\_error (*C++ function*), 37  
 cepton\_sdk::api::default\_on\_error (*C++ function*), 37  
 cepton\_sdk::api::get\_sensor\_information\_by\_serial\_number (*C++ function*), 38  
 cepton\_sdk::api::get\_sensor\_serial\_numbers (*C++ function*), 38  
 cepton\_sdk::api::get\_time (*C++ function*), 37  
 cepton\_sdk::api::has\_sensor\_by\_serial\_number (*C++ function*), 38  
 cepton\_sdk::api::initialize (*C++ function*), 37  
 cepton\_sdk::api::is\_end (*C++ function*), 37  
 cepton\_sdk::api::is\_live (*C++ function*), 37  
 cepton\_sdk::api::log\_error (*C++ function*), 37  
 cepton\_sdk::api::NetworkPacketCallback (*C++ class*), 38  
 cepton\_sdk::api::NetworkPacketCallback::NetworkPacketCallback (*C++ function*), 38  
 cepton\_sdk::api::NetworkPacketCallback::define\_initialize (*C++ function*), 38  
 cepton\_sdk::api::NetworkPacketCallback::initialize (*C++ function*), 38  
 cepton\_sdk::api::SensorErrorCallback (*C++ class*), 37  
 cepton\_sdk::api::SensorErrorCallback::global\_on\_callback (*C++ function*), 38  
 cepton\_sdk::api::SensorImageFrameCallback (*C++ class*), 38  
 cepton\_sdk::api::SensorImageFrameCallback::SensorImageFrameCallback (*C++ function*), 38  
 cepton\_sdk::api::SensorImageFrameCallback::define\_initialize (*C++ function*), 38  
 cepton\_sdk::api::SensorImageFrameCallback::initialize (*C++ function*), 38  
 cepton\_sdk::api::SensorImageFrameCallback::is\_initialized (*C++ function*), 38  
 cepton\_sdk::api::wait (*C++ function*), 37  
 cepton\_sdk::FpSensorErrorCallback (*C++ type*), 22  
 cepton\_sdk::SensorError (*C++ class*), 17  
 cepton\_sdk::SensorError::~SensorError (*C++ function*), 17  
 cepton\_sdk::SensorError::code (*C++ function*), 17  
 cepton\_sdk::SensorError::ignore (*C++ function*), 17  
 cepton\_sdk::SensorError::is\_error (*C++ function*), 17  
 cepton\_sdk::SensorError::is\_fault (*C++ function*), 17  
 cepton\_sdk::SensorError::msg (*C++ function*), 17  
 cepton\_sdk::SensorError::name (*C++ function*), 17  
 cepton\_sdk::SensorError::operator bool (*C++ function*), 17  
 cepton\_sdk::SensorError::operator SensorErrorCode (*C++ function*), 17  
 cepton\_sdk::SensorError::operator= (*C++ function*), 17  
 cepton\_sdk::SensorError::SensorError (*C++ function*), 17  
 cepton\_sdk::SensorError::used (*C++ function*), 17  
 cepton\_sdk::SensorHandle (*C++ type*), 25  
 cepton\_sdk::util::Callback (*C++ class*), 40  
 cepton\_sdk::util::Callback::clear (*C++ function*), 40  
 cepton\_sdk::util::Callback::global\_on\_callback (*C++ function*), 40  
 cepton\_sdk::util::Callback::listen (*C++ function*), 40  
 cepton\_sdk::util::Callback::operator () (*C++ function*), 40  
 cepton\_sdk::util::convert\_image\_point\_to\_point (*C++ function*), 39  
 cepton\_sdk::util::convert\_sensor\_image\_point\_to\_point (*C++ function*), 39

(C++ function), 40  
 cepton\_sdk::util::FrameAccumulator (C++ class), 41  
 cepton\_sdk::util::FrameAccumulator::add\_points (C++ function), 41  
 cepton\_sdk::util::FrameAccumulator::call\_back (C++ member), 41  
 cepton\_sdk::util::FrameAccumulator::clear (C++ function), 41  
 cepton\_sdk::util::FrameAccumulator::FrameAccumulator (C++ function), 41  
 cepton\_sdk::util::FrameAccumulator::get\_options (C++ function), 41  
 cepton\_sdk::util::FrameAccumulator::set\_options (C++ function), 41  
 cepton\_sdk::util::FrameDetector (C++ class), 41  
 cepton\_sdk::util::FrameDetector::add\_point (C++ function), 41  
 cepton\_sdk::util::FrameDetector::frame\_found (C++ member), 41  
 cepton\_sdk::util::FrameDetector::frame\_idx (C++ member), 41  
 cepton\_sdk::util::FrameDetector::frame\_x (C++ member), 41  
 cepton\_sdk::util::FrameDetector::FrameDetector (C++ member), 41  
 cepton\_sdk::util::FrameDetector::get\_options (C++ function), 41  
 cepton\_sdk::util::FrameDetector::reset (C++ function), 41  
 cepton\_sdk::util::FrameDetector::set\_options (C++ function), 41  
 cepton\_sdk::util::get\_timestamp\_usec (C++ function), 39  
 cepton\_sdk::util::OrganizedCloud (C++ class), 42  
 cepton\_sdk::util::OrganizedCloud::CellInfo (C++ class), 42  
 cepton\_sdk::util::OrganizedCloud::CellInfo::occ (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::CellInfo::orig (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::getIndex (C++ function), 42  
 cepton\_sdk::util::OrganizedCloud::height (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::info\_cells (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::n\_returns (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::points (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::timestamp\_end (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::timestamp\_start (C++ member), 42  
 cepton\_sdk::util::OrganizedCloud::width (C++ member), 42  
 cepton\_sdk::util::Organizer (C++ class), 43  
 cepton\_sdk::util::Organizer::binSize (C++ function), 43  
 cepton\_sdk::util::Organizer::CENTER (C++ enumerator), 43  
 cepton\_sdk::util::Organizer::mode (C++ function), 43  
 cepton\_sdk::util::Organizer::organize\_points (C++ function), 43  
 cepton\_sdk::util::Organizer::Organizer (C++ function), 43  
 cepton\_sdk::util::Organizer::OrganizerMode (C++ enum), 43  
 cepton\_sdk::util::Organizer::OrganizerSettings (C++ class), 44  
 cepton\_sdk::util::Organizer::OrganizerSettings::hor (C++ member), 44  
 cepton\_sdk::util::Organizer::OrganizerSettings::hor (C++ member), 44  
 cepton\_sdk::util::Organizer::OrganizerSettings::mod (C++ member), 44  
 cepton\_sdk::util::Organizer::OrganizerSettings::ver (C++ member), 44  
 cepton\_sdk::util::Organizer::OrganizerSettings::ver (C++ member), 44  
 cepton\_sdk::util::Organizer::RECENT (C++ enumerator), 43  
 cepton\_sdk::util::Organizer::settings (C++ function), 44  
 cepton\_sdk::util::SensorPoint (C++ class), 39  
 cepton\_sdk::util::SensorPoint::distance (C++ member), 39  
 cepton\_sdk::util::SensorPoint::flags (C++ member), 39  
 cepton\_sdk::util::SensorPoint::image\_x (C++ member), 39  
 cepton\_sdk::util::SensorPoint::image\_z (C++ member), 39  
 cepton\_sdk::util::SensorPoint::intensity (C++ member), 39  
 cepton\_sdk::util::SensorPoint::reserved (C++ member), 40  
 cepton\_sdk::util::SensorPoint::return\_type (C++ member), 40  
 cepton\_sdk::util::SensorPoint::saturated (C++ member), 40  
 cepton\_sdk::util::SensorPoint::timestamp (C++ member), 39

cepton\_sdk::util::SensorPoint::valid (C++ member), 39  
 cepton\_sdk::util::SensorPoint::x (C++ member), 40  
 cepton\_sdk::util::SensorPoint::y (C++ member), 40  
 cepton\_sdk::util::SensorPoint::z (C++ member), 40  
 cepton\_sdk::util::SensorPoint::[anonymous] (C++ member), 40  
 cepton\_sdk\_capture\_replay\_close (C function), 35  
 cepton\_sdk\_capture\_replay\_get\_enable\_loop (C function), 35  
 cepton\_sdk\_capture\_replay\_get\_filename (C function), 35  
 cepton\_sdk\_capture\_replay\_get\_length (C function), 35  
 cepton\_sdk\_capture\_replay\_get\_position (C function), 35  
 cepton\_sdk\_capture\_replay\_get\_speed (C function), 35  
 cepton\_sdk\_capture\_replay\_get\_start\_time (C function), 35  
 cepton\_sdk\_capture\_replay\_is\_end (C function), 35  
 cepton\_sdk\_capture\_replay\_is\_open (C function), 35  
 cepton\_sdk\_capture\_replay\_is\_running (C function), 36  
 cepton\_sdk\_capture\_replay\_open (C function), 35  
 cepton\_sdk\_capture\_replay\_pause (C function), 36  
 cepton\_sdk\_capture\_replay\_resume (C function), 36  
 cepton\_sdk\_capture\_replay\_resume\_blocking (C function), 35  
 cepton\_sdk\_capture\_replay\_resume\_blocking\_once (C function), 35  
 cepton\_sdk\_capture\_replay\_seek (C function), 35  
 cepton\_sdk\_capture\_replay\_set\_enable\_loop (C function), 35  
 cepton\_sdk\_capture\_replay\_set\_speed (C function), 35  
 cepton\_sdk\_clear (C function), 23  
 cepton\_sdk\_create\_frame\_options (C function), 22  
 cepton\_sdk\_create\_options (C function), 22  
 cepton\_sdk\_deinitialize (C function), 23  
 cepton\_sdk\_get\_control\_flags (C function), 23  
 cepton\_sdk\_get\_error (C function), 19  
 cepton\_sdk\_get\_frame\_length (C function), 23  
 cepton\_sdk\_get\_frame\_mode (C function), 23  
 cepton\_sdk\_get\_n\_sensors (C function), 27  
 cepton\_sdk\_get\_port (C function), 23  
 cepton\_sdk\_get\_sensor\_handle\_by\_serial\_number (C function), 27  
 cepton\_sdk\_get\_sensor\_information (C function), 27  
 cepton\_sdk\_get\_sensor\_information\_by\_index (C function), 27  
 cepton\_sdk\_has\_control\_flag (C function), 23  
 cepton\_sdk\_initialize (C function), 23  
 cepton\_sdk\_listen\_image\_frames (C function), 30  
 cepton\_sdk\_listen\_network\_packet (C function), 33  
 cepton\_sdk\_listen\_serial\_lines (C function), 31  
 cepton\_sdk\_set\_control\_flags (C function), 23  
 cepton\_sdk\_set\_frame\_options (C function), 23  
 cepton\_sdk\_set\_port (C function), 23  
 cepton\_sdk\_unlisten\_image\_frames (C function), 30  
 cepton\_sdk\_unlisten\_network\_packet (C function), 33  
 cepton\_sdk\_unlisten\_serial\_lines (C function), 31